M251 一顆風扇葉片上 RGB 燈的燈效控制

NuMicro® 32 位元系列微控制器範例程式碼介紹

文件資訊

應用簡述	本範例程式碼使用 M251 對一顆風扇葉片上 RGB 燈進行燈效控制
BSP 版本	M251_Series_BSP_CMSIS_V3.02.007
開發平台	NuMaker-M251SD-V1.1

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1. 概述

本範例程式碼使用 M251 對風扇一顆風扇葉片上 RGB 燈進行控制以得到不同的燈效。

風扇所使用的是扇葉上自帶三色 LED 燈的款式，該風扇主要用於顯示卡上的散熱單元，該風扇除了原本的散熱功能外，還可以透過不同燈效來增強視覺感受。顯卡上的針對燈效做的視覺效果可以大致分成三種，第一種是在風扇槽的外圈放置燈光，透過燈條實現燈光效果。第二種是將風扇本身視作一個燈罩，透過在軸心放置燈光來實現燈效。最後一種也是本範例程式介紹的，在風扇的其中一個葉片上鑲有 RGB LED 燈，透過風扇旋轉速度和 LED 燈的互相配合來達成燈光效果。

1.1 此範例程式會用到以下 MCU 功能來運作:

1. PWM: 用作直接控制 RGB LED 燈的顏色，因為有三個風扇，共需要九個輸出腳位。
2. QSPI: 用作模擬 PWM 訊號以控制三個風扇的轉速。
3. PDMA: 提高資料傳輸效率，以保證 QSPI 能維持傳輸速度。
4. Timer: 有兩個功能，Timer Capture 用於偵測風扇轉速，以校準燈效位置，Timer Time-out 用於更改 RGB LED 燈。
5. SysTick: 與 Timer 搭配用於更改 RGB LED 燈。
6. UART: 不用於風扇相關控制，用於 PC 端與 MCU 做資料傳輸。

1.2 RGB 燈風扇原理

該風扇上的 LED 燈使用紅、綠、藍(色光三原色)三路訊號最終結合為燈光呈現的顏色。此處控制燈光使用 PWM 訊號，利用短時間內不同佔空比的 PWM 訊號，用不同平均電壓的三原色 LED 燈混合來使燈光輸出想要的顏色。

除了混合 LED 燈外，還會用到人眼視覺暫留。人眼對於連續的燈有視覺暫留的現象，只要燈光更新的足夠快，實際非同一時間出現的燈光，在旋轉一整圈後和下一圈燈光重疊，就好像一整圈燈光同時持續在運作，不會有不連續的感覺。這也說明同一種燈效的燈光在相鄰數圈的出現位置需要一致，才能得到我們想要呈現的效果，不然燈光效果會混雜在一起造成達不到想要的效果。基於上述，燈光輸出需要配合風扇轉速來使燈光出現在想要的位置。

想要知道風扇的位置資訊，需要接收並分析風扇的 TACH 訊號。風扇旋轉時會回傳 TACH 訊號，該訊號連接上拉電阻(10k ohm)後連接於 TM_EXT 腳位。TACH 訊號每旋轉 90 度會發生一次訊號轉折，藉由這些訊號轉折可以知道風扇馬達在 0 度、90 度、180 度、270 度位置的時間點，也可以透過計算知道當前風扇的轉速。

風扇可以透過 PWM 的不同佔空比來控制風扇轉速。但即便給予相同的 PWM 訊號控制風扇轉速，風扇轉速也不會是恆定值，因此需要每旋轉一圈就更新一次當前的轉速值，可以用來讓燈光在指定位置顯示想要的顏色。若沒有持續更新風扇的轉速值，也會造成燈光沒辦法在指定位置顯示想要的顏色，造成燈光效果混亂。

有些需要散熱的設備上可能同時有三個風扇在運作。要想同時控制三個轉速不同的風扇實現燈光效果，需要節省 Timer 的使用。同時在一個 Timer 中開啟 Capture mode 和 Continuous counting mode 功能，可以做到一個 Timer 對應一個風扇的燈光控制。Timer capture mode 利用 TM_EXT(Timer external)腳位將 TACH 訊號做解析，判斷訊號轉折點並記錄相鄰兩個訊號轉折點的時間。利用風扇轉 90 度會發生一次 TACH 訊號轉折的時間來推算風扇的轉速。Timer 除了使用 Timer capture mode 外，還用 Continuous counting mode 控制燈光。要使用 Timer 控制燈光，需要知道如何配合 Capture mode 作為燈光控制的方式，圖 1-1 為燈效運作底層的說明。

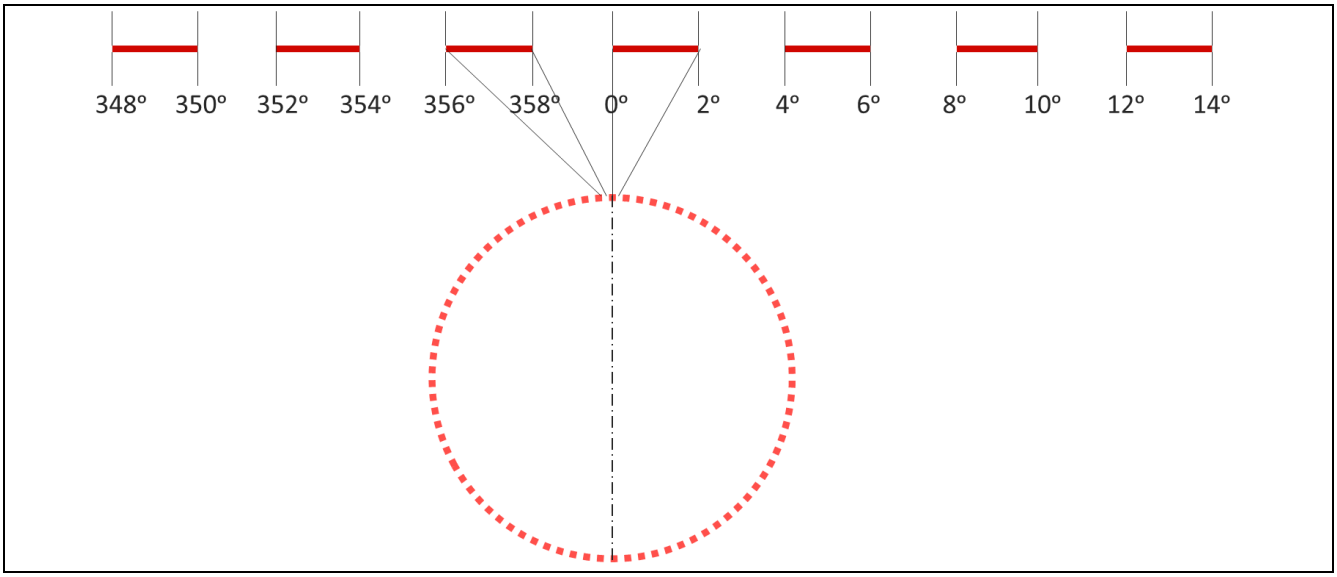


圖 1-1 風扇上燈效顯示原理

以每 2 度為一個單位做一段顯示效果(圖 1-1 使用紅白相間來表示)。換句話說，風扇每旋轉 2 度即進行一次燈光的顯示，用 Capture mode 計算的轉速可進一步計算旋轉 2 度花費的時間，此值可以設定 Continuous counting mode 的下一次 Time-out 值。用 Timer continuous counting

mode 使風扇每旋轉 2 度就進入一次 Timer 中斷更新燈光內容(因同一個 Timer 需要搭配 Timer capture mode 使用，所以不可使用 Periodic mode 控制燈光效果，使用 Periodic mode 會清除 TIMER_CNT(Timer counter) 暫存器的值，造成 Capture mode 值錯誤)，並更新 TIMER_CMP(Timer compare) 暫存器的值 (Continuous counting mode 下一次 Time-out 值) 以讓 Timer 持續動作。

在風扇旋轉完一圈後在 12 點鐘位置的燈光效果由 Timer capture mode 控制而不由 Timer continuous mode 的 Time-out 控制，在此點重新開始一圈的燈效顯示、更新風扇轉速、清除 TIF(Timer continuous counting mode 的 Time-out flag) 並更新 TIMER_CNT 值，這樣可以讓前後兩圈在 12 點鐘位置的燈光效果不會因為位置不同而產生混雜。

知道燈光的產生原理後即可按照想要的方式實現想要的效果。以下有三種燈光效果作為範例:

- 彩虹: 一整圈圓周為彩虹色，並往一個方向旋轉。
- 顏色流動: 隨著時間由一整圈的 A 顏色逐漸變成 B 顏色。
- 三爪:將圓分成三等分(每 120 度為一個區間)，其中每個區間只有前面 60 度發光，隔一段時間燈效往一個方向旋轉。

同一段程式會因為風扇的旋轉方向導致呈現出來的燈效會有所不同。為了方便說明，本文件關於燈效的說明統一用風扇順時針旋轉為範例進行圖片示例。

1.3 燈效原理-彩虹

此燈效為顯示一整圈的彩虹色，並且持續向一個方向進行旋轉，旋轉動作是持續進行的，計算平均每 500ms 會旋轉 90 度，如圖 1-2 所示。

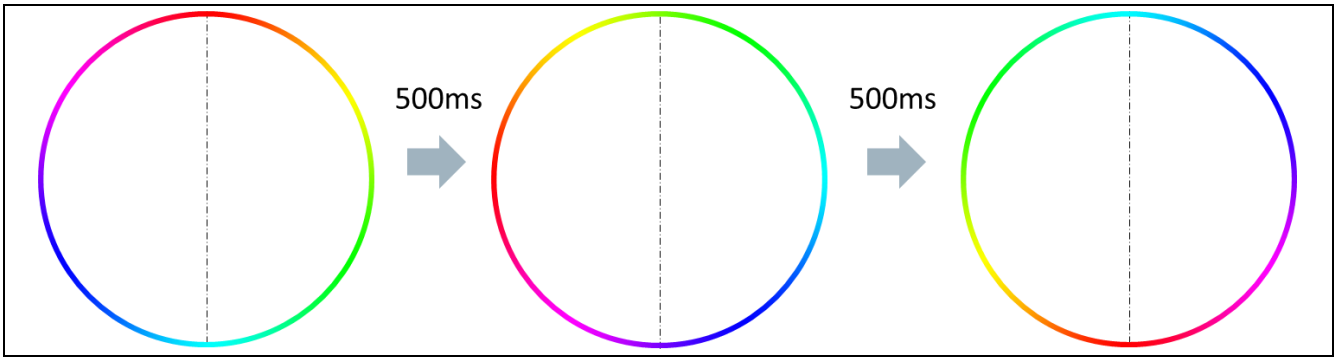


圖 1-2 彩虹燈效的圖片說明

為避免降低執行效率，在進入燈效控制前創建此燈效需要用到的陣列。因為顯示原理是以每 2 度為單位進行燈效更新，一個圓周會被切割為 180 份。因此前述陣列內容為 180 個 PWM duty 的元素，涵蓋了實現彩虹燈效的所有 PWM duty 值(在章節:程式碼介紹中呈現)。PWM 作為三色 LED 燈輸出在圓周中持續的循環，即構成了一整圈彩虹燈效。

在執行 Timer continuous counting mode 時每進入一次中斷就往陣列中下一個元素做為 PWM duty 的輸出，要使彩虹燈效持續旋轉而非每一圈彩虹都固定在圓周上的同一個位置，需要搭配 SysTick ，每旋轉一圈時檢查是否 SysTick 計時已到，若 SysTick 已經到達設定時間，將輸出起始的對應值加 1，這樣就會朝一個方向做旋轉的動作。

1.4 燈效原理-顏色流動

此燈效為顯示不同的整圈顏色，並且在中間顏色切換過渡時每 40ms 向旋轉方向增加 30 度的下一個顏色，共花費 480ms 來完成一次顏色的切換，如圖 1-3 所示。

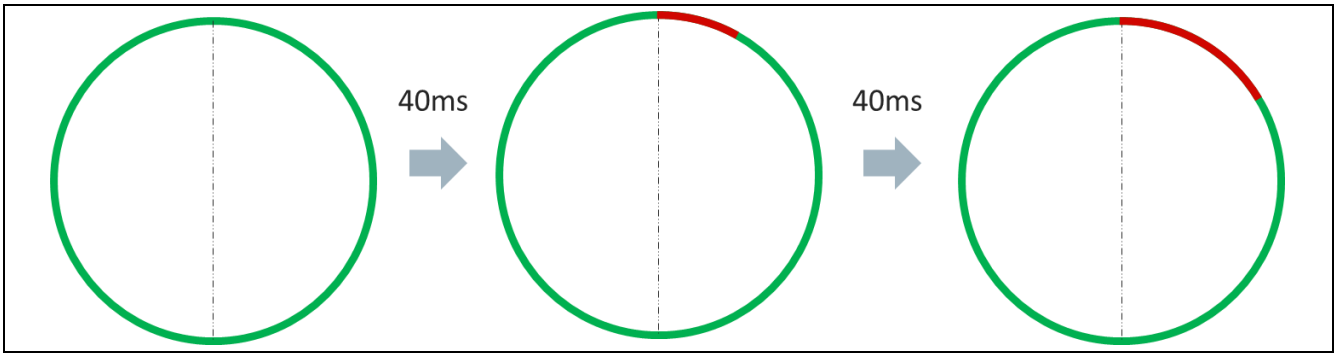


圖 1-3 顏色流動燈效的圖片說明

此圖為由綠色過渡到紅色的顯示，顏色朝一個方向。每次顏色切換完成後有 1.4 秒的時間維持同一個顏色，接著進行下一次的顏色過渡。此處需要搭配 SysTick clock 做顏色過渡和顯示同一個顏色的計時。

1.5 燈效原理-三爪

此燈效為一整圈有三個區域有顏色顯示，以十二點鐘方向為 0 度，分別為 0~60, 120~180, 240~300 度顯示顏色，並且每過 500ms 在圓周旋轉 40 度，共旋轉三次後即達成一次顯示循環，如圖 1-4 所示。

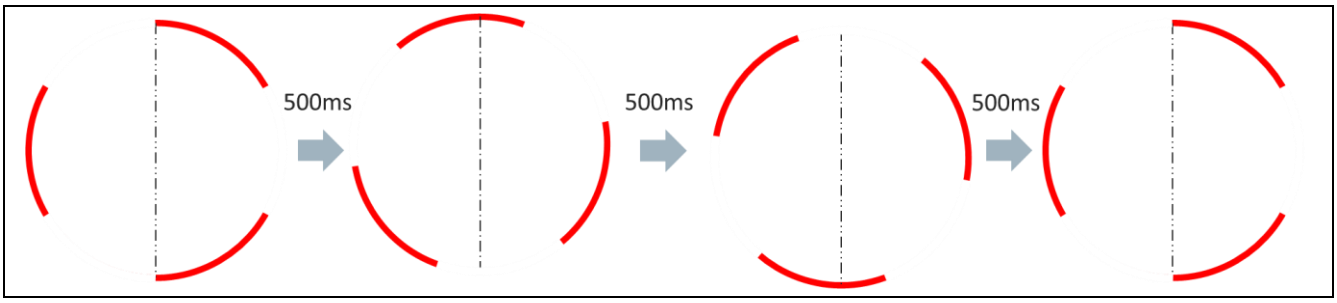


圖 1-4 三爪燈效的圖片說明(1)

在此之上加入色彩的轉換，每旋轉一次即更換一次顏色，如圖 1-5 所示。

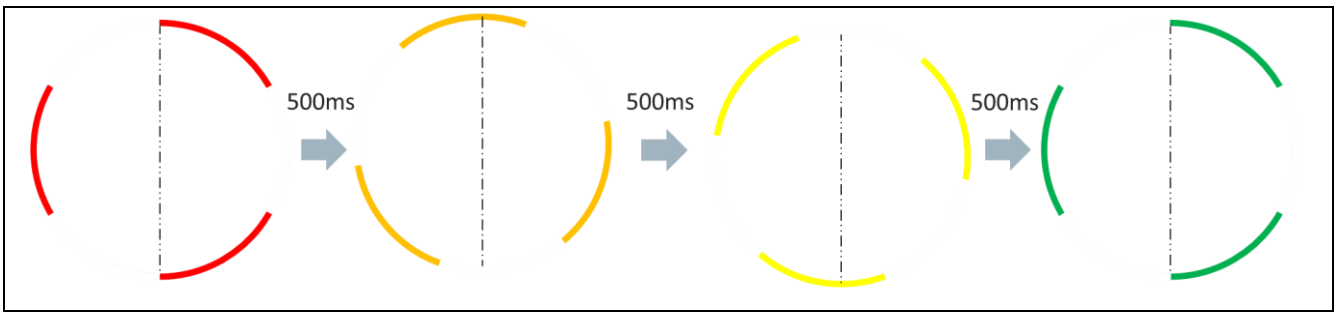


圖 1-5 三爪燈效的圖片說明(2)

1.6 執行結果

本範例程式碼執行編譯和燒錄的結果如圖 1-6 所示。

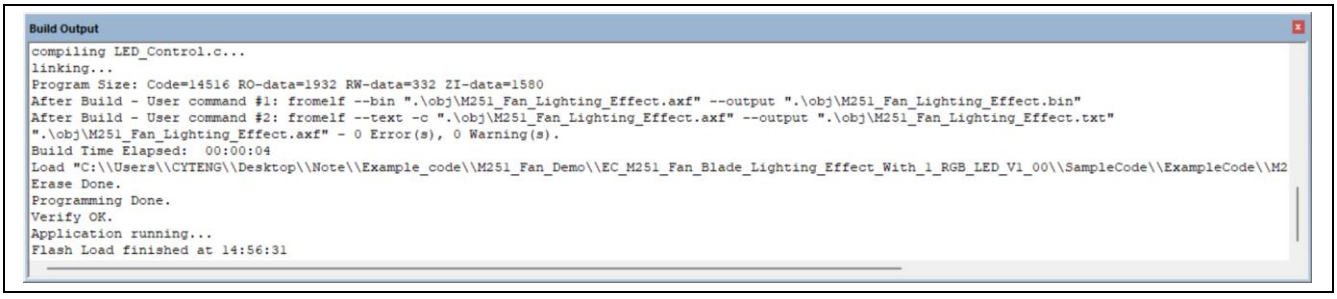


圖 1-6 執行編譯和燒錄結果

此風扇中央是順時針旋轉，左右兩側是逆時針旋轉，因此在執行時兩側燈效會不同於中央的風扇，會是相反方向，風扇照片如圖 1-7。

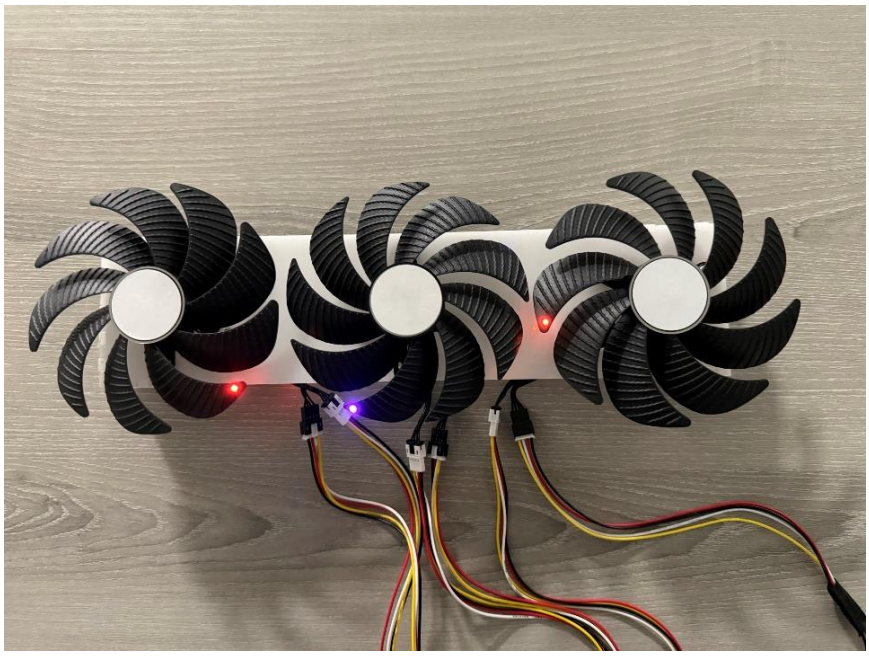


圖 1-7 風扇照片

此範例程式會在風扇旋轉的時候顯示燈效。然而相機拍攝時對影像的保存跟人眼視覺不相同，相機拍攝時會擷取一個極短時間的畫面資訊進行儲存，因此在拍攝風扇時，拍出來的影像會有燈效不滿一圈的情況，如圖 1-8。

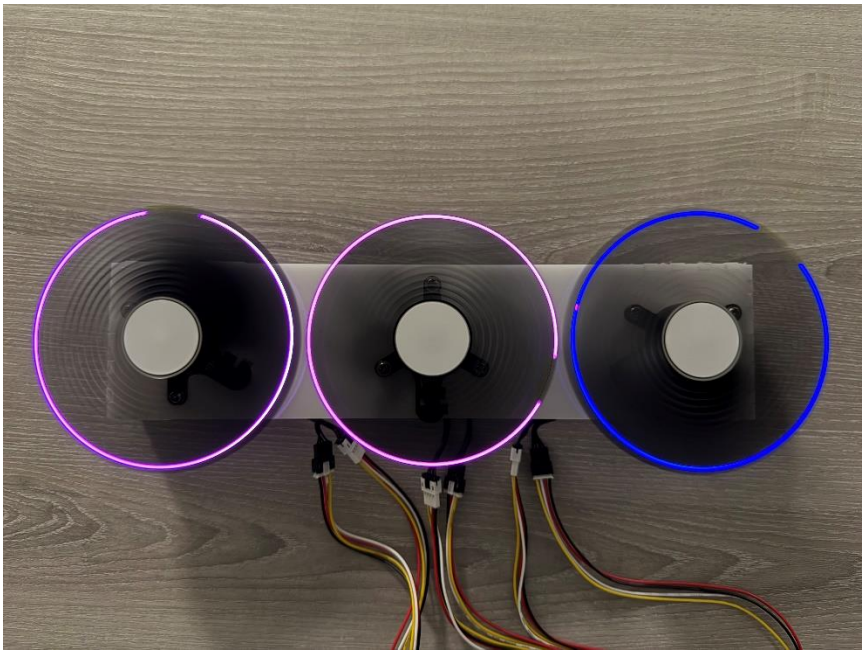


圖 1-8 一般拍攝時的相片

這時候可以利用增加曝光時間來改善拍攝的效果。執行結果如圖 1-9 至圖 1-13 所示。

雖然增加曝光時間可以改善拍攝的燈效，但是因三個風扇的轉速並不完全相同，轉速慢的風扇在旋轉完一圈時，轉速快的風扇已經多旋轉幾度了，這會造成相片在同個曝光時間下，有些風扇部分位置相鄰兩圈會發生顏色重疊，這個現象在拍攝彩虹燈效時可以明顯看出。

彩虹燈效如圖 1-9 所示。中央的風扇轉速較左右兩側快，造成中間風扇有白色出現。正常情況下想要顯示白色，可以讓三個顏色的 LED 燈用同個 PWM duty 輸出，在這裡拍攝出現了白色即代表相鄰兩圈的顏色在該位置出現了重疊。

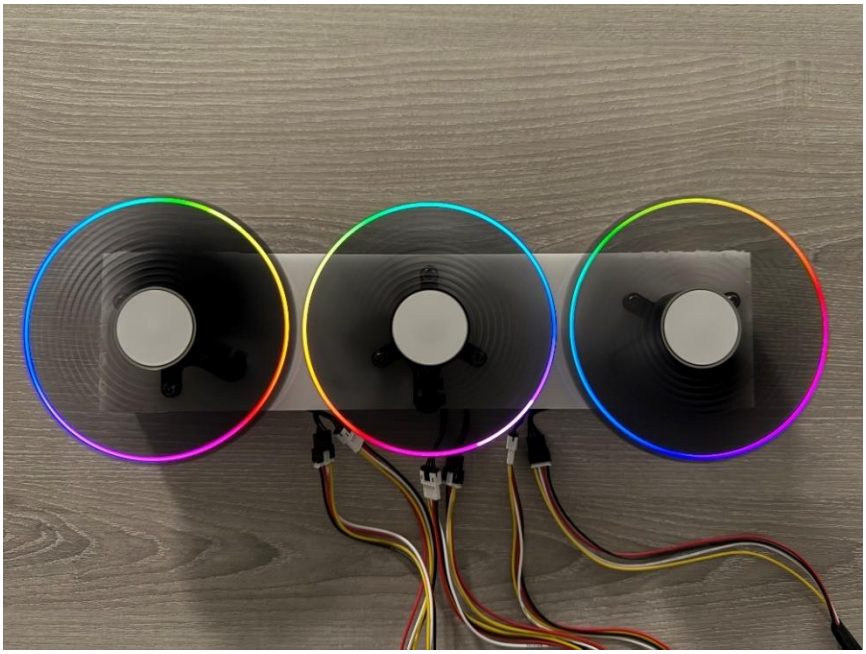


圖 1-9 彩虹燈效

顏色流動燈效如圖 1-10和圖 1-11所示。此為一次燈效改變所拍攝的前後對比照片。

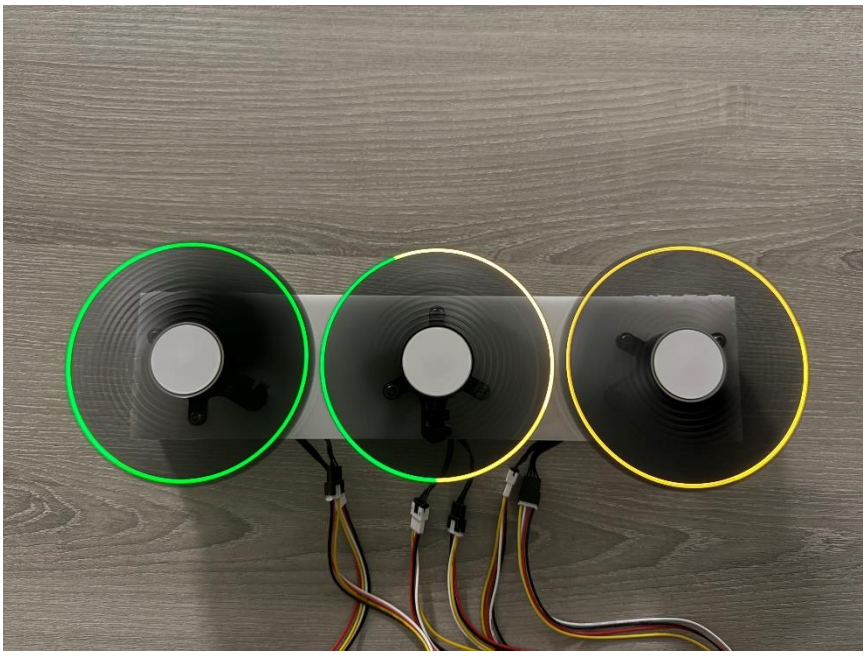


圖 1-10 顏色流動燈效(1)

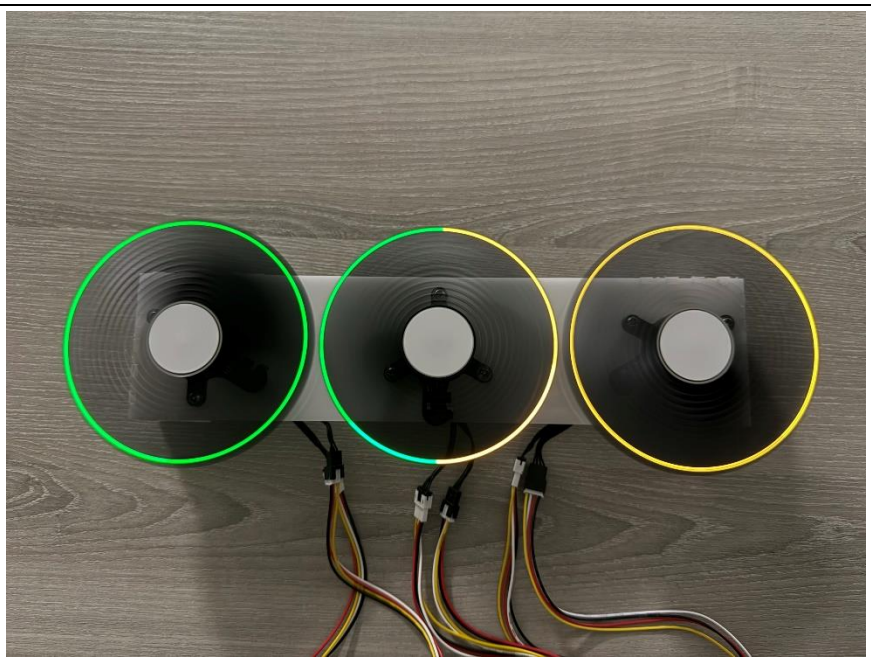


圖 1-11 顏色流動燈效(2)

三爪燈效如圖 1-12和圖 1-13所示。此為一次燈效改變所拍攝的前後對比照片。

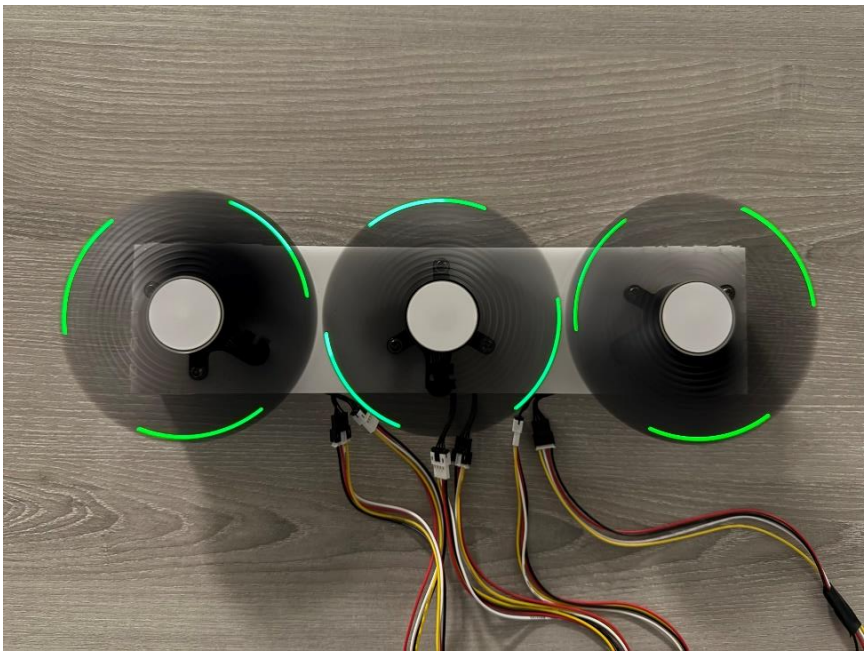


圖 1-12 三爪燈效(1)

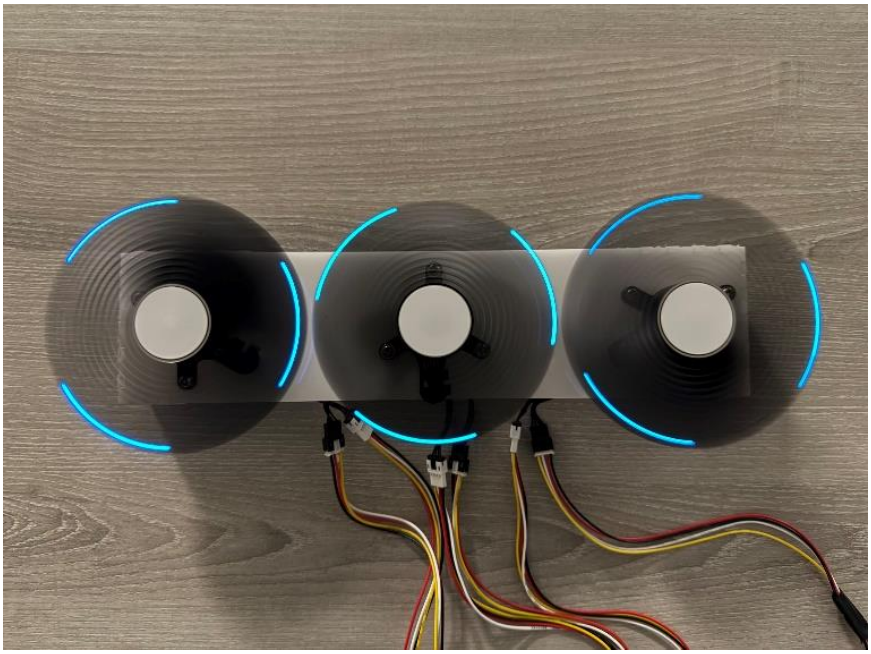


圖 1-13 三爪燈效(2)

2. 程式碼介紹

初始化系統後，對每個需要用到的周邊進行初始化的動作。程式碼位於 *main.c*。

```
int32_t main(void)
{
    /* Unlock protected registers */
    SYS_UnlockReg();

    /* Init System, IP clock and multi-function I/O */
    SYS_Init();

    /* Lock protected registers */
    SYS_LockReg();

    /* Init UART */
    UART0_Init();

    /* Init Timer0, Timer1, Timer2 */
    TIMER0_Init();
    TIMER1_Init();
    TIMER2_Init();

    /* Init All Timer Capture function & enable interrupt */
    TIMER_Capture_Init();

    /* Init Systick */
    SysTick_Initial();

    /* Start SysTick */
    SysTick->CTRL |= SysTick_CTRL_ENABLE_Msk;
```

函式 TIMER0_Init 內容，其他 Timer 的初始化內容相似，設定 Timer 為 Continuous mode，並且設定除頻器為 1(該數值 0 會在進行除頻時以(0+1)進行除頻動作)。

```
void TIMER0_Init()
{
    /* Set TIMER0 continuous time-out frequency is 1Hz */
    TIMER_Open(TIMER0, TIMER_CONTINUOUS_MODE, 1);

    /* Set TIMER0 prescale = 1 */
    TIMER_SET_PRESCALE_VALUE(TIMER0, 0);
}
```

函式 TIMER_Capture_Init 內容，Timer 在進行燈光控制同時也兼做風扇馬達 TACH 訊號的 Capture 動作，此函式即在做 Timer capture 和 Interrupt 啟用設定。由於 Timer 需要同時做 Capture 的動作，因此 Timer 必須使用 Continuous mode 來使 Timer 在 Time-out 發生後不重置計數器的值。

```
void TIMER_Capture_Init()
{
    //Enable Timer capture, capture interrupt, interrupt, NVIC
    TIMER_EnableCapture(TIMER0, TIMER_CAPTURE_FREE_COUNTING_MODE,
TIMER_CAPTURE_EVENT_RISING_FALLING);
    TIMER_EnableCaptureInt(TIMER0);
```

```
    TIMER_EnableInt(TIMER0);
    TIMER_EnableCapture(TIMER1, TIMER_CAPTURE_FREE_COUNTING_MODE,
TIMER_CAPTURE_EVENT_RISING_FALLING);
    TIMER_EnableCaptureInt(TIMER1);
    TIMER_EnableInt(TIMER1);
    TIMER_EnableCapture(TIMER2, TIMER_CAPTURE_FREE_COUNTING_MODE,
TIMER_CAPTURE_EVENT_RISING_FALLING);
    TIMER_EnableCaptureInt(TIMER2);
    TIMER_EnableInt(TIMER2);
    NVIC_EnableIRQ(TMR0_IRQn);
    NVIC_EnableIRQ(TMR1_IRQn);
    NVIC_EnableIRQ(TMR2_IRQn);
}
```

函式 SysTick_Initial 函式，對 SysTick 做初始化，動作包含選擇 Clock source、更新計數目標值(這個值會在燈效初始化時再被設定一次)、清除計數器值、啟用 SysTick 中斷。

```
void SysTick_Initial(void)
{
    /* Set SysTick clock source to HCLK */
    SysTick->CTRL = SysTick_CTRL_CLKSOURCE_Msk;

    /* Set load value to make SysTick period is 250ms */
    SysTick->LOAD = 48000000 / 4 ;

    /* Clear SysTick counter */
    SysTick->VAL = 0;

    /* Enable SysTick interrupt */
    SysTick->CTRL |= SysTick_CTRL_TICKINT_Msk;

    /* Enable SysTick NVIC */
    NVIC_EnableIRQ(SysTick_IRQn);
}
```

接續 main 內容，為了控制風扇的轉速，需要 MCU 提供一個 PWM 訊號來控制風扇轉速。此程式使用 QSPI 來模擬 PWM 訊號。要使用 QSPI 模擬 PWM 訊號，首先須建立風扇轉速的 PWM duty 陣列以提供 QSPI 輸出訊號的來源，並使用 PDMA(Peripheral Direct Memory Access)來進行資料傳輸。如果要理解詳細原理，可以在 NuForum 參考另一個 Nuvoton Example code "[M251] Using QSPI with PDMA to Simulate PWM" 來了解其原理以及應用，在此範例中，利用 QSPI 模擬的 PWM 訊號佔空比值固定為 50%。相似的程式碼在此不再重述。

```
    /* Setting PDMA Structure for QSPI */
    /* DSCT0 Description*/
    DMA_DESC_QSPI[0].u32Ctl =
        ((40 - 1) << PDMA_DSCT_CTL_TXCNT_Pos) | /* Transfer count is 40 */ \
        PDMA_WIDTH_32 |   /* Transfer width is 32 bits(one word) */ \
        PDMA_SAR_INC |    /* Source increment size is 32 bits(one word) */ \
        PDMA_DAR_FIX |    /* Destination increment size is 32 bits(one word) */ \
        PDMA_REQ_SINGLE |  /* Transfer type is burst transfer type */ \
        PDMA_BURST_128 |  /* Burst size is 128. No effect in single transfer type */ \
        PDMA_TBINTDIS_DISABLE |   /* Disable transfer done and table empty interrupt */ \
        PDMA_OP_SCATTER;  /* Operation mode is scatter-gather mode */
```

```
    /* Configure source address */
    DMA_DESC_QSPI[0].u32Src = (uint32_t)Duty_array;

    /* Configure destination address */
    DMA_DESC_QSPI[0].u32Dest = (uint32_t)&QSPI0->TX;

    /* Configure next descriptor table address */
    DMA_DESC_QSPI[0].u32Offset = (uint32_t)&DMA_DESC_QSPI[1] - (PDMA->SCATBA); /* next
descriptor table is table 2 */
    /* next descriptor table is table 2 */

    /* DSCT1 Description*/
    DMA_DESC_QSPI[1].u32Ctl =
        ((40 - 1) << PDMA_DSCT_CTL_TXCNT_Pos) | /* Transfer count is 40 */ \
        PDMA_WIDTH_32 |   /* Transfer width is 32 bits(one word) */ \
        PDMA_SAR_INC |    /* Source increment size is 32 bits(one word) */ \
        PDMA_DAR_FIX |    /* Destination increment size is 32 bits(one word) */ \
        PDMA_REQ_SINGLE |  /* Transfer type is burst transfer type */ \
        PDMA_BURST_128 |  /* Burst size is 128. No effect in single transfer type */ \
        PDMA_TBINTDIS_DISABLE |   /* Disable transfer done and table empty interrupt */ \
        PDMA_OP_SCATTER;  /* Operation mode is scatter-gather mode */

    /* Configure source address */
    DMA_DESC_QSPI[1].u32Src = (uint32_t)Duty_array;

    /* Configure destination address */
    DMA_DESC_QSPI[1].u32Dest = (uint32_t)&QSPI0->TX;

    /* Configure next descriptor table address */
    DMA_DESC_QSPI[1].u32Offset = (uint32_t)&DMA_DESC_QSPI[0] - (PDMA->SCATBA); /* next
descriptor table is table 1 */
    /* next descriptor table is table 1 */

    /* Calculate PWM output duty array */
    Calculate_Duty_array(Duty_right_now);

    /* Init QSPI & PDMA, Start QSPI output */
    QSPI_and_PDMA_Init();
```

接續 main 內容，初始化控制 LED 燈的 PWM，並創建彩虹燈效會用到的陣列，陣列內容為跑一整輪彩虹效果需要用到的全部 PWM duty 值。並且啟動 Timer 和 PWM、指定三個風扇的燈效並做燈效初始化動作。

```
    /* Init PWM output */
    PWM_Init();

    /* Create Rainbow lighting effect array */
    for (uint8_t i = 0; i < 180; i++)
    {
        if (i < 40)
        {
            Rainbow_array[i] = 100;
        }
        else if ((i > 39) && (i < 80))
        {
            Rainbow_array[i] = 100 - 2.5 * (i - 40);
        }
        else if ((i > 79) && (i < 140))
        {
            Rainbow_array[i] = 0;
```

```
        }
        else if ((i > 139) && (i < 180))
        {
            Rainbow_array[i] = 2.5 * (i - 140);
        }
    }

    //Timer start
    TIMER_Start(TIMER0);
    TIMER_Start(TIMER1);
    TIMER_Start(TIMER2);

    /* Start PWM counter */
    PWM_Start(fan_left.PWM_generator, fan_left.PWM_RMSK);
    PWM_Start(fan_left.PWM_generator, fan_left.PWM_GMSK);
    PWM_Start(fan_left.PWM_generator, fan_left.PWM_BMSK);
    PWM_Start(fan_middle.PWM_generator, fan_middle.PWM_RMSK);
    PWM_Start(fan_middle.PWM_generator, fan_middle.PWM_GMSK);
    PWM_Start(fan_middle.PWM_generator, fan_middle.PWM_BMSK);
    PWM_Start(fan_right.PWM_generator, fan_right.PWM_RMSK);
    PWM_Start(fan_right.PWM_generator, fan_right.PWM_GMSK);
    PWM_Start(fan_right.PWM_generator, fan_right.PWM_BMSK);

    /* Init lighting effect: Rainbow lighting mode */
    Lighting_Mode_Setting = RAINBOW_LIGHTING_MODE;
    fan_left.light_mode_choose = Lighting_Mode_Setting ;
    fan_middle.light_mode_choose = Lighting_Mode_Setting ;
    fan_right.light_mode_choose = Lighting_Mode_Setting ;
    Rainbow_initial(&fan_left);
    Rainbow_initial(&fan_middle);
    Rainbow_initial(&fan_right);
```

根據上述所創建的 Rainbow_array 表如下，在彩虹燈效套用該陣列時，紅色的 Duty 值為 Rainbow_array[i]，綠色的 Duty 值為 Rainbow_array[i+60]，藍色的 Duty 值為 Rainbow_array[i+120]，套用陣列的元素項若大於 179 則取對 180 的餘數。這樣就變成表 2-1 對照表，剛好在經過 180 個元素會變成一輪彩虹顏色的循環。

i	0	20	40	60	80	100	120	140	160
Rainbow_array[i] (R)	100	100	100	50	0	0	0	0	50
G	0	0	50	100	100	100	50	0	0
B	50	0	0	0	0	50	100	100	100

表 2-1 Rainbow_array 對照表

函式 PWM_Init 內容，設定 PWM 的初始頻率與 Duty 值，並將輸出訊號做反向(Inverse)方便後續計算。請注意，建議 PWM 頻率要高於風扇旋轉 1 度時間去換算的頻率，原因為若 PWM 頻

率不夠高，風扇在旋轉時 PWM 控制的燈光不夠跑完一輪週期，會造成燈光顯示分層，達不到想要的燈光效果。

舉例說明，欲在 0 度~2 度的位置顯示燈光紫色(R,G,B)=(128,0,255)，若 PWM 輸出頻率換算的週期為風扇馬達旋轉 2 度的時間，由 0 度位置開始新一輪的 PWM 輸出，該燈光會在 0 度~1 度顯示亮紫色(R,G,B)=(255,0,255)，在 1 度~2 度顯示藍色(R,G,B)=(0,0,255)，這即是因為 PWM 頻率不夠快所導致的結果，如圖 2-1。

```
void PWM_Init()
{
    //For PWM Initial Setting
    PWM_ConfigOutputChannel(fan_left.PWM_generator, fan_left.PWM_Rchannel, 50000,
100);//PWM0 ch0
    PWM_ConfigOutputChannel(fan_left.PWM_generator, fan_left.PWM_Gchannel, 50000,
100);//PWM0 ch1
    PWM_ConfigOutputChannel(fan_left.PWM_generator, fan_left.PWM_Bchannel, 50000,
100);//PWM0 ch2
    PWM_ConfigOutputChannel(fan_middle.PWM_generator, fan_middle.PWM_Rchannel, 50000,
100);//PWM0 ch3
    PWM_ConfigOutputChannel(fan_middle.PWM_generator, fan_middle.PWM_Gchannel, 50000,
100);//PWM0 ch4
    PWM_ConfigOutputChannel(fan_middle.PWM_generator, fan_middle.PWM_Bchannel, 50000,
100);//PWM0 ch5
    PWM_ConfigOutputChannel(fan_right.PWM_generator, fan_right.PWM_Rchannel, 50000,
100);//PWM1 ch3
    PWM_ConfigOutputChannel(fan_right.PWM_generator, fan_right.PWM_Gchannel, 50000,
100);//PWM1 ch4
    PWM_ConfigOutputChannel(fan_right.PWM_generator, fan_right.PWM_Bchannel, 50000,
100);//PWM1 ch5

    /* Enable output of PWM0, PWM1 */
    PWM_EnableOutput(fan_left.PWM_generator, fan_left.PWM_RMSK);
    PWM_EnableOutput(fan_left.PWM_generator, fan_left.PWM_GMSK);
    PWM_EnableOutput(fan_left.PWM_generator, fan_left.PWM_BMSK);
    PWM_EnableOutput(fan_middle.PWM_generator, fan_middle.PWM_RMSK);
    PWM_EnableOutput(fan_middle.PWM_generator, fan_middle.PWM_GMSK);
    PWM_EnableOutput(fan_middle.PWM_generator, fan_middle.PWM_BMSK);
    PWM_EnableOutput(fan_right.PWM_generator, fan_right.PWM_RMSK);
    PWM_EnableOutput(fan_right.PWM_generator, fan_right.PWM_GMSK);
    PWM_EnableOutput(fan_right.PWM_generator, fan_right.PWM_BMSK);

    //PWM Inverse
    PWM0->POLCTL |= 0x3FUL; //ch0~ch5
    PWM1->POLCTL |= 0x38UL; //ch3~ch5
}
```

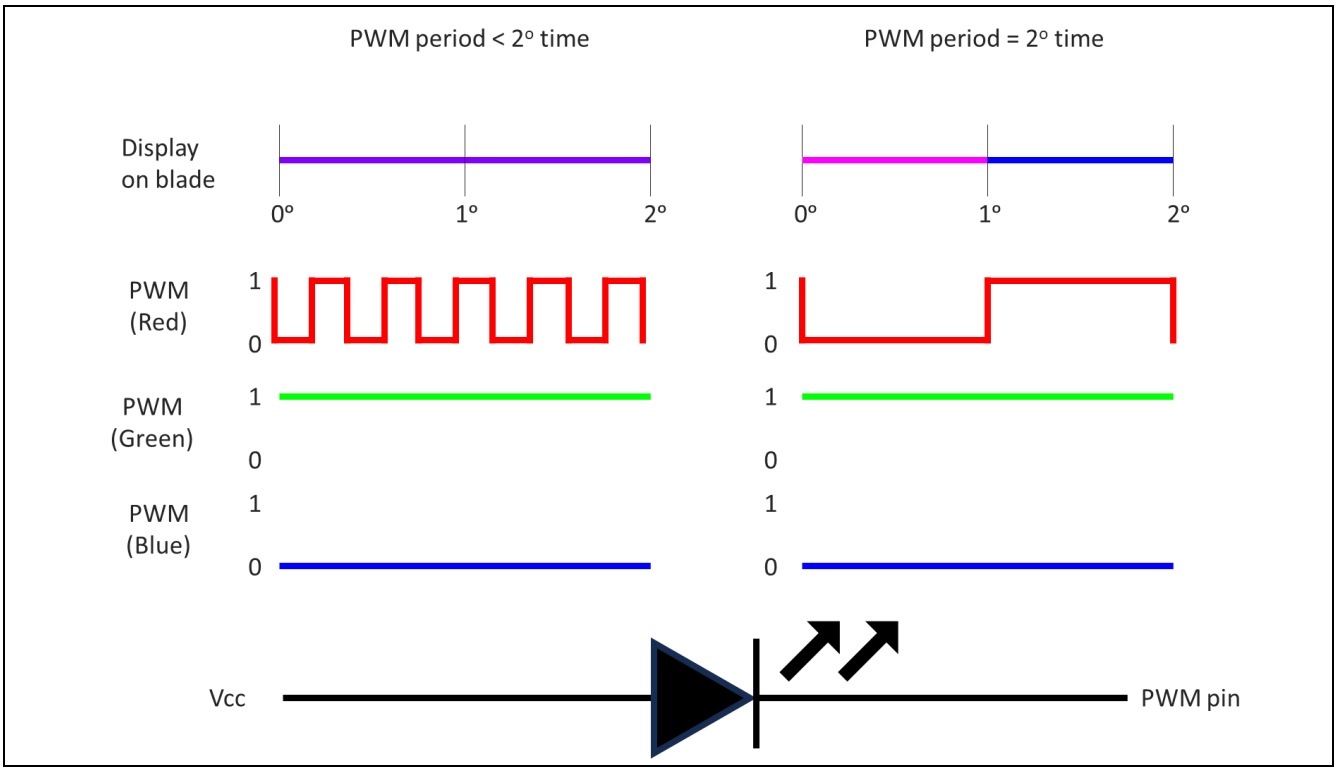


圖 2-1 PWM 頻率對燈效影響說明

其他對燈效初始化的函式，例如 Rainbow_initial 函式是關於切換不同燈效時需要將一些變數初始化，以確保切換前的燈效變數不會影響到接下來的燈效。函式一開始的 PWM_EnableOutput 是因為三爪燈效可能會關閉當前 PWM 輸出，因此需要在初始化的時候做一個固定開啟的動作。最後是切換 SysTick 的目標值並重置 SysTick 重新開始計時。

```
void Rainbow_initial(struct fan* fan_setting)
{
    PWM_EnableOutput(fan_setting->PWM_generator, fan_setting->PWM_RMSK);
    PWM_EnableOutput(fan_setting->PWM_generator, fan_setting->PWM_GMSK);
    PWM_EnableOutput(fan_setting->PWM_generator, fan_setting->PWM_BMSK);

    /* Init variable*/
    fan_setting->update_count = fan_setting->Capture_Init_count_Offset;
    fan_setting->fan_count = 0;
    fan_setting->count_initial = 0;
    fan_setting->R_duty = 0;
    fan_setting->G_duty = 0;
    fan_setting->B_duty = 0;
    fan_setting->systick_flag = 0;

    /* Setting SysTick time-out value */
    SysTick->LOAD = RAINBOW_SYSTICK_LOAD_VALUE * fan_setting->Speed;

    /* Reset SysTick */
    SysTick->VAL = 0;
}
```

這邊再列出 Color_shift_initial 函式內容，這邊相較 Rainbow_initial 函式多了將顏色值填入 Color_R_Duty 等三色的動作，這是因為不同於 Rainbow 燈效是用 Rainbow_array，這邊是用 Default_Color 陣列去當作過渡的顏色去計算燈效，並且將 RGB 的 0~255 值轉換為 PWM Duty 的 0~100 值(後面的亮度可以按照使用者想要的亮度去調整結構體的內容，正常使用時不需要去更動)。Color_Transition 變數是用於表示這個風扇此時是處於變化前的顏色、或是正在處於顏色過渡的過程，三個風扇的 Color_Transition 初始值不一樣可以讓三個風扇有順序的進行燈效。

```
void Color_shift_initial(struct fan* fan_setting)
{
    PWM_EnableOutput(fan_setting->PWM_generator, fan_setting->PWM_RMSK);
    PWM_EnableOutput(fan_setting->PWM_generator, fan_setting->PWM_GMSK);
    PWM_EnableOutput(fan_setting->PWM_generator, fan_setting->PWM_BMSK);

    /* Init Basic Color */
    for (uint8_t i = 0; i < fan_setting->Color_Num; i++)
    {
        fan_setting->Color_R_Duty[i] =  Default_Color[i][0] * DUTY_MAX / COLOR_VALUE_MAX *
fan_setting->Brightness / BRIGHTNESS_MAX;
        fan_setting->Color_G_Duty[i] =  Default_Color[i][1] * DUTY_MAX / COLOR_VALUE_MAX *
fan_setting->Brightness / BRIGHTNESS_MAX;
        fan_setting->Color_B_Duty[i] =  Default_Color[i][2] * DUTY_MAX / COLOR_VALUE_MAX *
fan_setting->Brightness / BRIGHTNESS_MAX;
    }

    /* Init variable*/
    fan_left.Color_Transition = 2;
    fan_middle.Color_Transition = 1;
    fan_right.Color_Transition = 0;
    fan_setting->update_count = fan_setting->Capture_Init_count_Offset;
    fan_setting->fan_count = 0;
    fan_setting->count_initial = 0;
    fan_setting->color_select = 0;
    fan_setting->change_count = 0;
    fan_setting->systick_flag = 0;

    /* Setting SysTick time-out value */
    SysTick->LOAD = COLOR_SHIFT_SYSTICK_LOAD_VALUE * fan_setting->Speed;

    /* Reset SysTick */
    SysTick->VAL = 0;
}
```

在介紹燈效之前，需要先說明一下 SysTick 中斷和 Timer 中斷內容，兩個中斷都與控制燈效有著密切的關係，前者是用來幫助計時切換燈效到下一幀的時間，後者則是負責對風扇馬達的 TACH 進行 Capture 和在風扇上每個位置需要輸出控制 LED 燈的 PWM 訊號。

SysTick 中斷函式內容，每進來兩次中斷函式就將切換訊號的 Flag 設為 1，這些 Flag 會在執行燈效的時候被清除。因為每個風扇轉速不一樣，flag 不應該三個風扇共用。基於上述原因，不

建議在燈效函式中直接檢查 SysTick 的 Time-out flag，這樣 flag 被清除時可能會有風扇沒切換到燈效。

```
void SysTick_Handler(void)
{
    /* Reset SysTick */
    SysTick->VAL = 0;

    /* Set flag = 1 when enter SysTick interrupt twice */
    if (SysTick_count > 1)
    {
        fan_left.systick_flag = 1;
        fan_middle.systick_flag = 1;
        fan_right.systick_flag = 1;
        SysTick_count = 0;
    }

    /* Calculate SysTick count */
    SysTick_count++;
}
```

Timer 中斷函式內容，這裡以 TMR0_IRQHandler 舉例，Timer1 和 Timer2 的中斷程式同理。首先檢查 Capture flag，該 Flag 用以偵測風扇馬達在運作時回傳的 TACH 訊號，Capture 此訊號有兩個作用，一個是藉由此訊號可以得知風扇當前的轉速，燈效更新會利用風扇轉速作為時間依據來做更新。第二個是藉由此訊號可以得知風扇現在旋轉到的"相對位置"，因為風扇馬達每次在上電啟動時起始位置不一樣，因此 TACH 開始產生的位置也會不一樣，沒有辦法用 TACH 訊號來表示風扇馬達的絕對位置。得知風扇的相對位置後記錄進入 Capture 中斷次數可以得知現在風扇是否旋轉了一圈。

在旋轉了一圈後，更新風扇的轉速來給下次 Timer 的燈效更新做參考，與讓以一圈為單位進行循環的燈效有一個起始參考點，除了讓我們在計算燈效的參數上更方便外，也讓風扇馬達旋轉中相鄰兩圈的燈效不至於差太多影響最後燈效的呈現。結尾清除可能發生的 Time-out 中斷 Flag。

```
void TMR0_IRQHandler(void)
{
    /* Check capture flag=1, otherwise is time out interrupt occur */
    if (TIMER_GetCaptureIntFlag(TIMER0) == 1)
    {
        /* Clear capture interrupt flag */
        TIMER_ClearCaptureIntFlag(TIMER0);

        /* Calculate capture count, if perform four times capture, it means the fan motor
has completed one revolution.  */
        fan_left.Capture_Init_count++;

        /* Record the capture position when the motor starts rotating after being powered
on.*/
```

```
        fan_left.Capture_Init_count = fan_left.Capture_Init_count % (4 +
fan_left.Capture_Init_count_Start_position) + fan_left.Capture_Init_count_Start_position;

        /* If the fan motor has completed one revolution. */
        if ((fan_left.update_count >  3) && (fan_left.Capture_Init_count ==
+fan_left.Capture_Init_count_Offset))
        {
            /* Update fan speed */
            Update_Timer_Capture(&fan_left);

            /* Initial fan_count & determine PWM output  */
            Timer_capture_function(&fan_left);

            /* Init update count(capture count) */
            fan_left.update_count = 0;

            /* Setting next Timer time-out value */
            TIMER0->CMP = TIMER0->CNT + fan_left.CalCMP[3];

            /* Clear time out interrupt flag(TIF) to prevent re-entering the interrupt
immediately after exiting the interrupt  */
            TIMER_ClearIntFlag(TIMER0);
            NVIC_ClearPendingIRQ(TMR0_IRQn);

        }

        /* Calculate update count(capture count) */
        fan_left.update_count++;
    }
```

Update_Timer_Capture 函式內容，透過相鄰兩次旋轉一圈的 Timer capture 值的差值來得出實際的計數值，該值即為旋轉 360 度需要的時間，後續燈效以旋轉 2 度為一次更新，因此將該值除以 180 並儲存，用以後續計算 Continuous mode 下次 Time-out 時間。

```
void Update_Timer_Capture(struct fan* fan_setting)
{
    //Put t-1 count value in CalCMP[1],  Scan Capture and put data in CalCMP[0],
Difference value Calculate in CalCMP[2]
    fan_setting->CalCMP[1] = fan_setting->CalCMP[0];
    fan_setting->CalCMP[0] = TIMER_GetCaptureData(fan_setting->Timer_Generator);
    fan_setting->CalCMP[2] = fan_setting->CalCMP[0] - fan_setting->CalCMP[1];
    fan_setting->CalCMP[2] = fan_setting->CalCMP[2] & 0xffffff;
    fan_setting->CalCMP[3]=(fan_setting->CalCMP[2]/180)& 0xffffff;
}
```

接續 Timer 中斷函式內容。在沒有 Capture flag 時進入 Timer 中斷表示風扇旋轉了 2 度，需要切換燈效，這時進入燈運作函式 Timer_Light_active_function，變更 LED 燈顯示。結束變更 LED 燈後，由於 Timer 為 Continuous mode，需要重新設定 TIMER_CMP 值，以讓 Timer 知道下次進來中斷的時間(風扇再旋轉 2 度後的時間)。圖 2-2 說明了一整圈圓周上觸發 Timer 中斷的類型。

```
    /* Time out interrupt */
    else if ((TIMER_GetCaptureIntFlag(TIMER0) == 0))
    {
        /* Clear time out interrupt flag */
        TIMER_ClearIntFlag(TIMER0);

        /* Active lighting effect */
        Timer_Light_active_function(&fan_left);

        /* Setting next Timer time-out value */
        TIMER0->CMP = TIMER0->CMP + fan_left.CalCMP[3];
    }
}
```

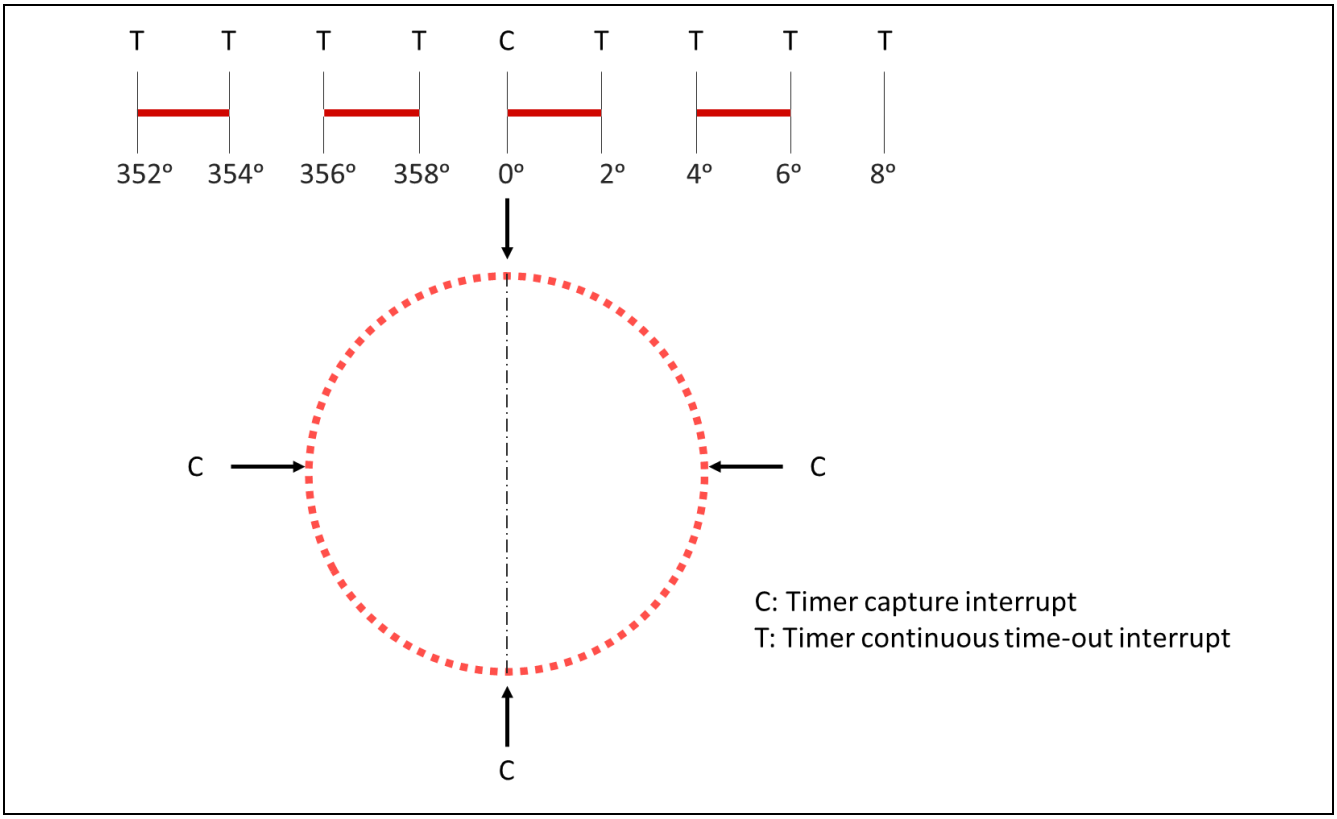


圖 2-2 圓周與 Timer Interrupt 說明

最後是位於 *LED_Control.c* 中的燈效函式，這些函式在上述的 Timer 中斷程式裡面會用到，分成兩個部分，一個是在上述風扇旋轉一圈的更新燈效，另一個是風扇旋轉 2 度後的更新燈效。

在風扇旋轉一圈後，使用 Timer_capture_function 函式將用來持續記錄風扇燈效的主要變數:fan_count 做一圈參考點的重置，這樣可以不因相鄰兩圈的 fan_count 值不一樣造成燈效嚴重飄動(若沒有做參考點的重置)。

在顏色流動中，除了做參考點的重置外，還有做第一次自參考點重置完的第一次燈效顯示。Color_Transition 變數如同 Color_shift_initial 函式的介紹，在該變數值等於 3 時表示正處於顏色過渡中，在重置點時需要顯示變化後的顏色;在該變數值小於 3 時表示處於變化前的顏色，在重置點時需要顯示變化前的顏色。

```
void Timer_capture_function(struct fan* fan_setting)
{
    switch (fan_setting->light_mode_choose)
    {
        case RAINBOW_LIGHTING_MODE:
        {
            /* Reset fan_count at capture point */
            fan_setting->fan_count = fan_setting->count_initial;
        }
        break;

        case COLOR_SHIFT_LIGHTING_MODE:
        {
            /* Determine the value of the capture point based on the value of
Color_Transition */
            if (fan_setting->Color_Transition == 3)
            {
                fan_setting->u32NewCMR_R = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Rchannel, fan_setting->Color_R_Duty[(fan_setting->color_select + 1) %
fan_setting->Color_Num], 100);
                fan_setting->u32NewCMR_G = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Gchannel, fan_setting->Color_G_Duty[(fan_setting->color_select + 1) %
fan_setting->Color_Num], 100);
                fan_setting->u32NewCMR_B = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Bchannel, fan_setting->Color_B_Duty[(fan_setting->color_select + 1) %
fan_setting->Color_Num], 100);
            }
            else if (fan_setting->Color_Transition < 3)
            {
                fan_setting->u32NewCMR_R = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Rchannel, fan_setting->Color_R_Duty[fan_setting->color_select ], 100);
                fan_setting->u32NewCMR_G = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Gchannel, fan_setting->Color_G_Duty[fan_setting->color_select ], 100);
                fan_setting->u32NewCMR_B = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Bchannel, fan_setting->Color_B_Duty[fan_setting->color_select ], 100);
            }

            /* Reset fan_count at capture point */
            fan_setting->count_initial = 0;
            fan_setting->fan_count = fan_setting->count_initial;
        }
        break;

        case CLAWS_LIGHTING_MODE:
        {
            /* Reset fan_count at capture point */
            fan_setting->fan_count = fan_setting->count_initial;
        }
        break;
```

```
    }

    /* Setting comparator value, to change PWM duty */
    PWM_SET_CMR(fan_setting->PWM_generator, fan_setting->PWM_Rchannel, fan_setting-
>u32NewCMR_R);
    PWM_SET_CMR(fan_setting->PWM_generator, fan_setting->PWM_Gchannel, fan_setting-
>u32NewCMR_G);
    PWM_SET_CMR(fan_setting->PWM_generator, fan_setting->PWM_Bchannel, fan_setting-
>u32NewCMR_B);
}
```

Timer_Light_active_function 函式內容。該函式為大部分進入 Timer 中斷時要使用的函數，根據當前為何種燈效來進入對應的計算，接下來會一個一個進行說明。

彩虹燈效內容。在 SysTick 發生 Time-out 後，更改 count_initial 的值用以讓進入 Timer_capture_function 後修改 fan_count 為該值。接下來根據 fan_count 值(這邊紅色、綠色、藍色的值各不相同)，選擇對應的 Rainbow_array 陣列的元素作為 PWM Duty，即可在對應的位置顯示，一整個圓周連起來即為彩虹色，如圖 2-3 所示。

```
void Timer_Light_active_function(struct fan* fan_setting)
{
    switch (fan_setting->light_mode_choose)
    {
        case RAINBOW_LIGHTING_MODE:
        {
            /* When systick_flag =1, change next lighting effect frame */
            if (fan_setting->systick_flag == 1) // flag 0->1 : 11ms
            {
                fan_setting->systick_flag = 0;
                fan_setting->count_initial++;
                fan_setting->count_initial = fan_setting->count_initial % (360 /
COUNT_ANGLE_UNIT);
            }

            /* Change PWM output each time when enter interrupt */
            fan_setting->R_duty = Rainbow_array[fan_setting->fan_count];
            fan_setting->G_duty = Rainbow_array[fan_setting->fan_count_G];
            fan_setting->B_duty = Rainbow_array[fan_setting->fan_count_B];
            fan_setting->u32NewCMR_R = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Rchannel, fan_setting->R_duty, 100);
            fan_setting->u32NewCMR_G = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Gchannel, fan_setting->G_duty, 100);
            fan_setting->u32NewCMR_B = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Bchannel, fan_setting->B_duty, 100);

            fan_setting->fan_count++;
            fan_setting->fan_count = fan_setting->fan_count % (COUNT_ANGLE_360);

            /* Update fan_count_G & fan_count_B value */
            fan_setting->fan_count_G = fan_setting->fan_count + 120;
            fan_setting->fan_count_G = fan_setting->fan_count_G % (COUNT_ANGLE_360);
            fan_setting->fan_count_B = fan_setting->fan_count + 60;
            fan_setting->fan_count_B = fan_setting->fan_count_B % (COUNT_ANGLE_360);
```

```
        }
        break;
    }
```

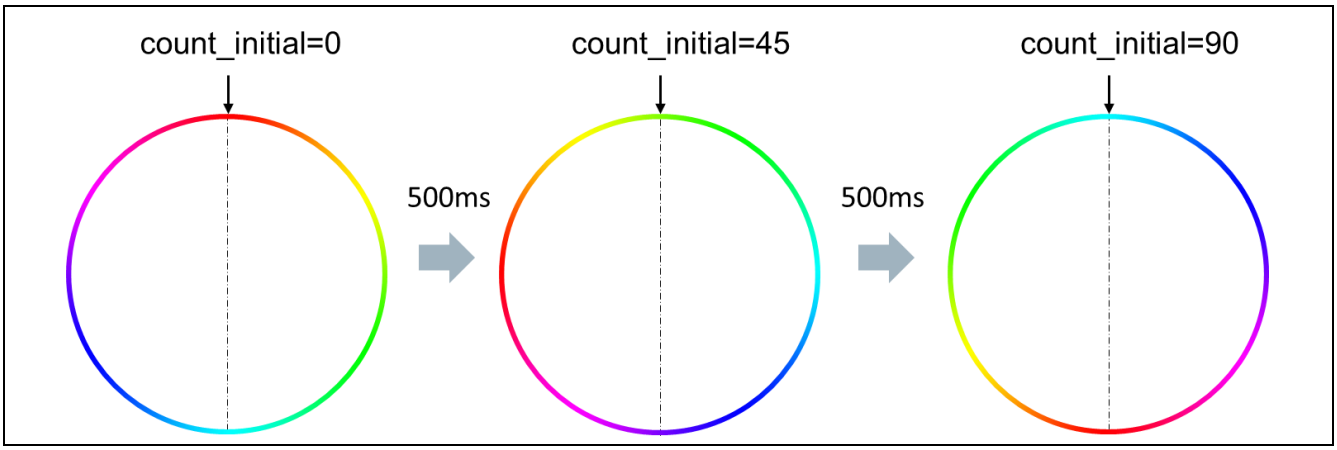


圖 2-3 彩虹燈效說明

顏色流動燈效內容，在 SysTick 發生 Time-out 後，更改 change_count 的值用以接下來計算和 Timer_capture_function 的計算。change_count 為每一圈燈效進行時顏色 1 轉為顏色 2 的點，因為顏色流動中是每隔一段時間才發生一次顏色的往前過渡，所以要由 change_count 來記錄現在已經過渡到圓周中哪個位置。這樣連續的進行燈效人眼看起來就好像逐漸由顏色 1 轉換到顏色 2。Color_Transition 如同 Timer_capture_function 的介紹。

在進行顏色過渡時(Color_Transition=3)根據 change_count 來判斷需要輸出顏色 1 還是顏色 2。其餘時間(Color_Transition<3)則輸出顏色 1。

進行顏色過渡時(Color_Transition=3)如圖 2-4 所示，change_count 以 15(COUNT_ANGLE_30)為一個單位增加，表示每一次 SysTick 到 Time-out 時增加 30 度，這樣進行一圈就能完成一次燈效。

```
        case COLOR_SHIFT_LIGHTING_MODE:
        {
            /* When systick_flag =1, change next lighting effect frame */
            if (fan_setting->systick_flag == 1)
            {
                fan_setting->systick_flag = 0;
                fan_setting->change_count = fan_setting->change_count + (COUNT_ANGLE_30);

                if (fan_setting->change_count > (COUNT_ANGLE_30 * 11))
                {
                    fan_setting->change_count = 0;

                    if (fan_setting->Color_Transition < 3)
```

```
                    {
                        fan_setting->Color_Transition++;
                    }
                    else
                    {
                        fan_setting->color_select = (fan_setting->color_select + 1) %
fan_setting->Color_Num;
                        fan_setting->Color_Transition = 0;
                    }
                }
            }

            /* Set the lighting effect according to Color_Transition */
            if (fan_setting->Color_Transition == 3)
            {
                if (fan_setting->fan_count < fan_setting->change_count)
                {
                    fan_setting->u32NewCMR_R = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Rchannel, fan_setting->Color_R_Duty[(fan_setting->color_select + 1) %
fan_setting->Color_Num], 100);
                    fan_setting->u32NewCMR_G = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Gchannel, fan_setting->Color_G_Duty[(fan_setting->color_select + 1) %
fan_setting->Color_Num], 100);
                    fan_setting->u32NewCMR_B = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Bchannel, fan_setting->Color_B_Duty[(fan_setting->color_select + 1) %
fan_setting->Color_Num], 100);
                }
                else if (fan_setting->fan_count > (fan_setting->change_count - 1))
                {

                    fan_setting->u32NewCMR_R = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Rchannel, fan_setting->Color_R_Duty[fan_setting->color_select], 100);
                    fan_setting->u32NewCMR_G = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Gchannel, fan_setting->Color_G_Duty[fan_setting->color_select], 100);
                    fan_setting->u32NewCMR_B = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Bchannel, fan_setting->Color_B_Duty[fan_setting->color_select], 100);

                }
            }
            else if (fan_setting->Color_Transition < 3)
            {
                fan_setting->u32NewCMR_R = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Rchannel, fan_setting->Color_R_Duty[fan_setting->color_select], 100);
                fan_setting->u32NewCMR_G = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Gchannel, fan_setting->Color_G_Duty[fan_setting->color_select], 100);
                fan_setting->u32NewCMR_B = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Bchannel, fan_setting->Color_B_Duty[fan_setting->color_select], 100);
            }

            fan_setting->fan_count++;
            fan_setting->fan_count = fan_setting->fan_count % (COUNT_ANGLE_360);
        }
        break;
```

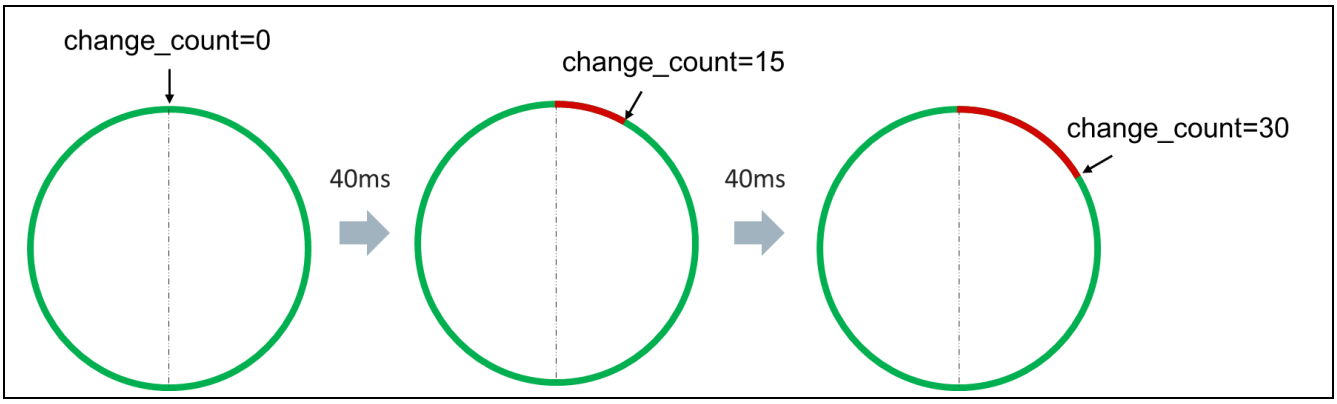


圖 2-4 顏色流動燈效說明

三爪燈效內容，在 SysTick 發生 Time-out 後，更改 count_initial 的值用以讓進入 Timer_capture_function 後修改 fan_count 為該值。再來修改 LED 燈顯示的顏色至下一個顏色，這邊用 color_select 表應顯示陣列內的哪個顏色，Color_Num 為該陣列總共有多少顏色，在 color_select 超過陣列內顏色時將 color_select 歸零。

然後就是呈現三爪的效果，利用在特定區域取消 PWM 輸出和特定區域啟用 PWM 輸出來達成三爪的效果。舉例說明，圖 2-5 中最左邊 count_initial=0，由 12 點鐘方向順時針每 60 度變更一次亮暗。在 count_initial=20 時，整個燈效往逆時針方向移動 40 度。在 count_initial=40 時，整個燈效再往逆時針方向移動 40 度。三次即可達成一次燈效在位置上的循環。

顏色與燈效位置上的搭配即達成此燈效的顯示。

```
        case CLAWS_LIGHTING_MODE:
        {
            /* When systick_flag =1, change next lighting effect frame */
            if (fan_setting->systick_flag == 1)
            {
                fan_setting->systick_flag = 0;
                fan_setting->count_initial = (fan_setting->count_initial + 20) % 60;

                fan_setting->color_select = (fan_setting->color_select + 1) %
fan_setting->Color_Num;

                fan_setting->u32NewCMR_R = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Rchannel, fan_setting->Color_R_Duty[fan_setting->color_select], 100);
                fan_setting->u32NewCMR_G = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Gchannel, fan_setting->Color_G_Duty[fan_setting->color_select], 100);
                fan_setting->u32NewCMR_B = CalNewDutyCMR(fan_setting->PWM_generator,
fan_setting->PWM_Bchannel, fan_setting->Color_B_Duty[fan_setting->color_select], 100);


                PWM_SET_CMR(fan_setting->PWM_generator, fan_setting->PWM_Rchannel,
fan_setting->u32NewCMR_R);
                PWM_SET_CMR(fan_setting->PWM_generator, fan_setting->PWM_Gchannel,
fan_setting->u32NewCMR_G);
```

```
                PWM_SET_CMR(fan_setting->PWM_generator, fan_setting->PWM_Bchannel,
fan_setting->u32NewCMR_B);
            }

            //Claws bright zone in first frame(angle): 0~60  120~180, 240~300
            if ((fan_setting->fan_count < COUNT_ANGLE_60) | (((COUNT_ANGLE_120 - 1) <
fan_setting->fan_count) & (fan_setting->fan_count < COUNT_ANGLE_180)) | \
                            (((COUNT_ANGLE_240 - 1) < fan_setting->fan_count) &
(fan_setting->fan_count < COUNT_ANGLE_300)))
            {
                PWM_EnableOutput(fan_setting->PWM_generator, fan_setting->PWM_RMSK);
                PWM_EnableOutput(fan_setting->PWM_generator, fan_setting->PWM_GMSK);
                PWM_EnableOutput(fan_setting->PWM_generator, fan_setting->PWM_BMSK);

            }
            //Claws dark zone in first frame(angle): 60~120, 180~240, 300~360
            else if ((((COUNT_ANGLE_60 - 1) < fan_setting->fan_count) &
(fan_setting->fan_count < COUNT_ANGLE_120)) | \
                            (((COUNT_ANGLE_180 - 1) < fan_setting->fan_count) &
(fan_setting->fan_count < COUNT_ANGLE_240)) | (((COUNT_ANGLE_300 - 1) <
fan_setting->fan_count) & (fan_setting->fan_count < COUNT_ANGLE_360)))
            {
                PWM_DisableOutput(fan_setting->PWM_generator, fan_setting->PWM_RMSK);
                PWM_DisableOutput(fan_setting->PWM_generator, fan_setting->PWM_GMSK);
                PWM_DisableOutput(fan_setting->PWM_generator, fan_setting->PWM_BMSK);
            }
            fan_setting->fan_count++;
            fan_setting->fan_count = fan_setting->fan_count % (COUNT_ANGLE_360);
        }
        break;
```

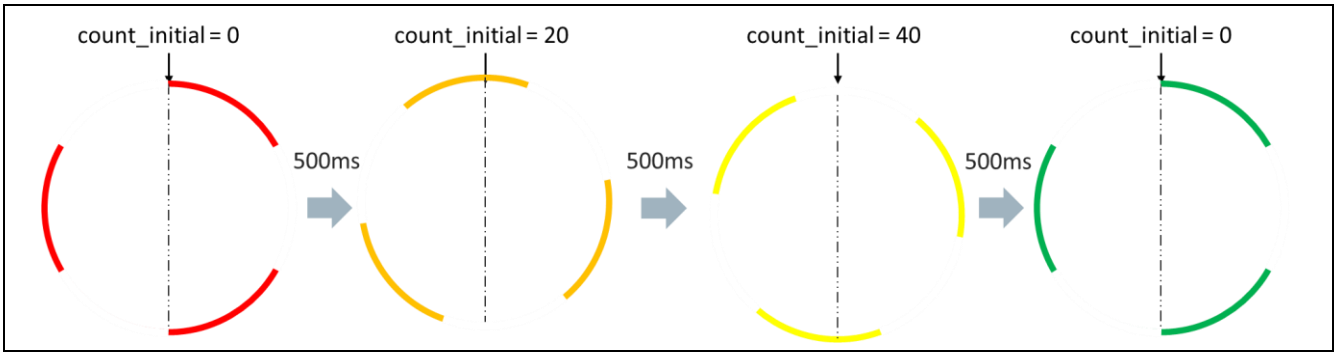


圖 2-5 三爪燈效說明

3. 軟體與硬體需求

3.1 軟體需求

- BSP 版本
  - M251_Series_BSP_CMSIS_V3.02.007
- IDE 版本
  - Keil uVersion 5.39

3.2 硬體需求

- 電路元件
  - NuMaker-M251SD V1.1
- 其他
  - 三個帶有 LED 燈的風扇
- 線路連接
  - 將 UART0 TX (PB.13) pin 腳連接到 PC UART RX、 UART0 RX (PB.12) pin 腳連接到 PC UART TX，並開啟 Nu-Link2-Me 上的 TXD 和 RXD switch 以顯示範例程式碼的執行結果。
  - 一個風扇有數條線，每一個風扇的腳位都需要接到指定腳位，風扇數量有限的話，也可以接小於 3 個風扇。

	Fan 1 pin	Fan 2 pin	Fan 3 pin
TACH(FG)	PA.11	PA.10	PA.9
PWM(QSPI MOSI)	PA.0	PA.0	PA.0
LED_R	PB.5	PB.2	PC.2
LED_G	PB.4	PB.1	PC.1
LED_B	PB.3	PB.0	PC.0

  - TACH 需要上拉電阻以讓風扇能正確收到 Capture 訊號

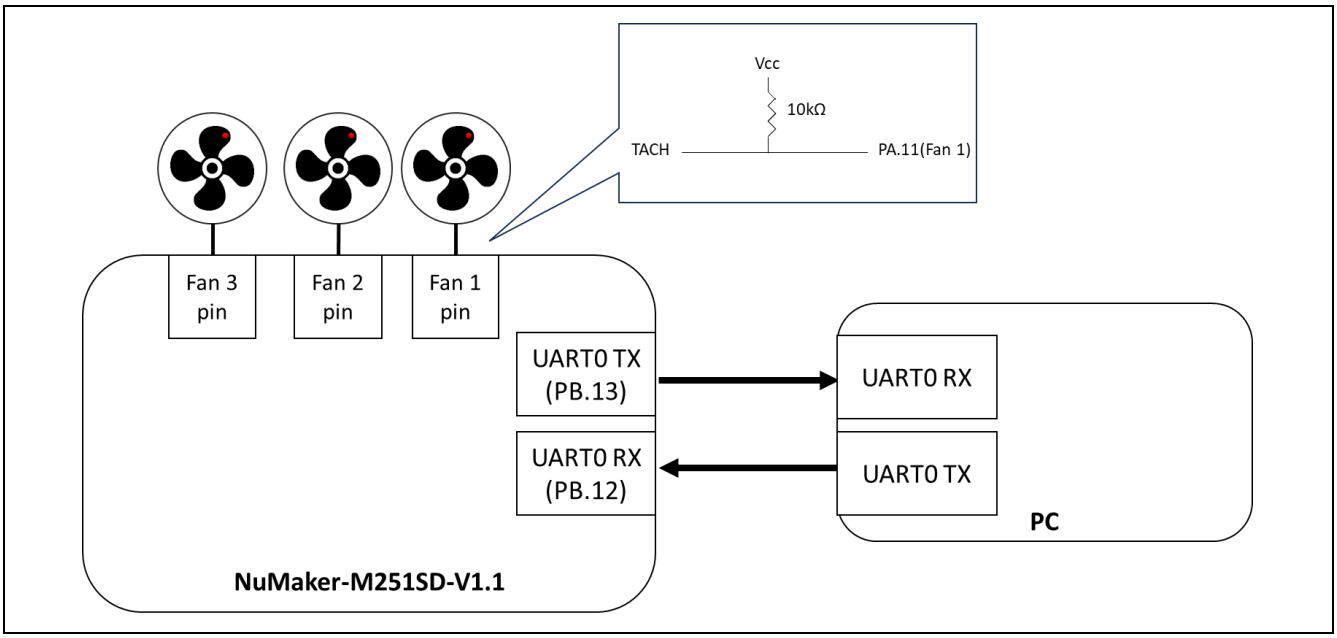

Vcc
10kΩ
TACH
PA.11(Fan 1)
Fan 3 pin
Fan 2 pin
Fan 1 pin
UART0 TX (PB.13)
UART0 RX (PB.12)
UART0 RX
UART0 TX
PC
NuMaker-M251SD-V1.1

圖 3-1 線路示意圖

4. 目錄資訊

目錄結構如圖 4-1 所示。

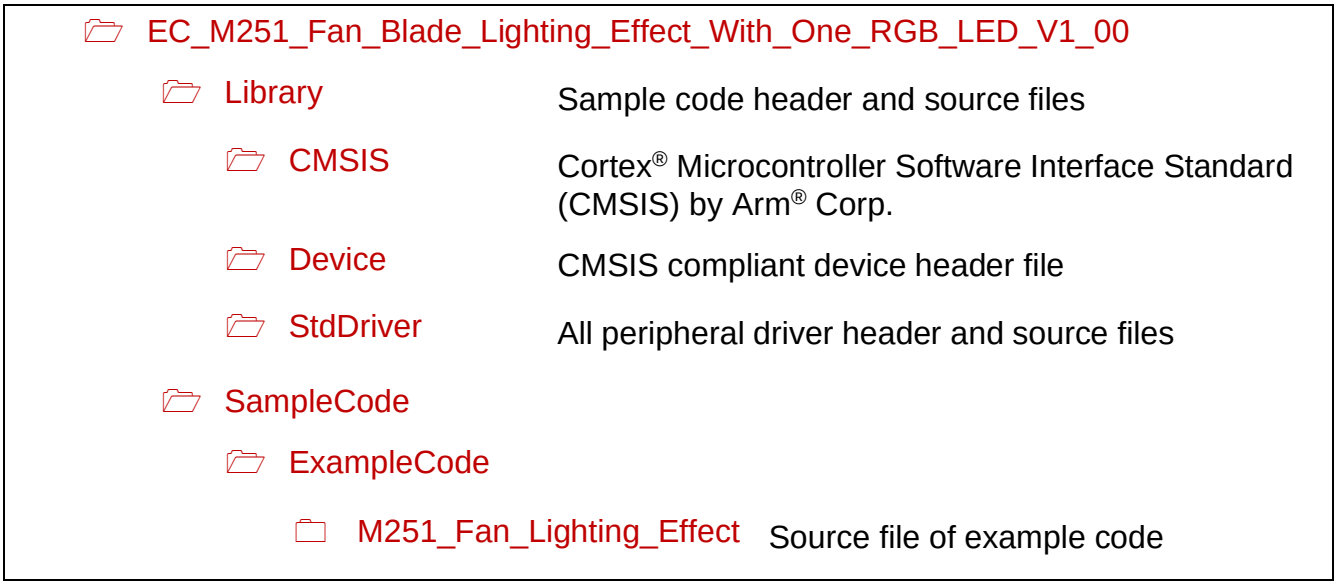


圖 4-1 目錄資訊

5. 範例程式執行

1. 根據目錄資訊章節進入 ExampleCode 路徑中的 KEIL 資料夾，雙擊 *M251_Fan_Lighting_Effect.uvprojx*。
2. 進入編譯模式介面
   - 編譯
   - 下載程式碼至記憶體
   - 進入 / 離開除錯模式
3. 進入除錯模式介面
   - 執行程式碼
4. 使用終端機程式(ex: Tera Term)，根據 UART 回傳的訊息發送指令來切換燈效。

6. 修訂紀錄

Date	Revision	Description
2025.07.01	1.00	初始發布。

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.